Available online at www.sciencedirect.com
ScienceDirect
Science of Computer Programming 66 (2007) 60–70
Science of Computer Programming
www.elsevier.com/locate/scico

Software requirements and architecture modeling for evolving non-secure applications into secure applications

Michael E. Shin[a,*], Hassan Gomaa[b]

[a] Department of Computer Science, Texas Tech University, Lubbock, TX 79409-3104, United States
[b] Department of Information and Software Engineering, George Mason University, Fairfax, VA 22030-4444, United States

Received 13 March 2006; received in revised form 1 September 2006; accepted 13 October 2006
Available online 25 January 2007

Abstract

This paper describes an approach to modeling the evolution of non-secure applications into secure applications in terms of the software requirements model and software architecture model. The requirements for security services are captured separately from application requirements, and the security services are encapsulated in connectors in the software architecture, separately from the components providing functional services. The enterprise architecture is described in terms of use case models, static models, and dynamic models. The software architecture is described in terms of components and connectors, which can be deployed to distributed configurations. By separating application concerns from security concerns, the evolution from a non-secure application to a secure application can be achieved with less impact on the application. An electronic commerce system is described to illustrate the approach.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Evolution; Security; Application system; Software requirements; Software architecture

1. Introduction

Many applications have been developed without considering security services such as confidentiality, integrity, access control, and non-repudiation. As software security is becoming more important in business these days, non-secure applications need to evolve to become secure applications. Several approaches have been suggested to address this problem [1,2,12,14,16,19,26], but these approaches mainly focus on the treatment of security in a specific phase of software development, such as requirements modeling or software architecture. Therefore, it is necessary for a systematic approach to evolve a non-secure application into a secure application.

In this paper, the enterprise architecture is described in terms of use case models, static models, and dynamic models. Furthermore, the software architecture is described in terms of components and connectors, which can be deployed to distributed configurations. In the evolution of a non-secure application to a secure application, security concerns in the software requirements model and software architecture need to be separated from application concerns, so that the application can be more maintainable. Security requirements are often considered to be non-functional

* Corresponding author. Tel.: +1 806 7423527.
E-mail addresses: Michael.Shin@ttu.edu (M.E. Shin), hgomaa@gmu.edu (H. Gomaa).

requirements, but they lead to functional requirements for realizing security services [16,27]. These security functions need to be modeled separately from application functions, in both the requirements model and software architecture model, as a non-secure application evolves into a secure application.

This paper describes how to model the evolution of a non-secure application to a secure application in terms of the requirements model and software architecture model, which are both described using the UML notation [3,21]. By careful separation of concerns in the software requirements model, security requirements for security services are modeled distinctly from application requirements. The security services modeled in the requirements model are encapsulated in connectors separately from components in the software architecture. Separation of security concerns from application concerns makes a secure application more evolvable and maintainable. It should be pointed out that this paper addresses software application security. It is assumed that operating system and network security mechanisms (for the platforms on which the secure applications execute) are already provided to address security vulnerabilities exploited by viruses, worms, hackers, and other malicious intrusions, and denial of service attacks.

This paper begins by describing the work related to this paper in Section 2. Section 3 describes the overview of our approach. Section 4 describes the evolution of the requirements model for secure applications. Section 5 describes evolving to a secure software architecture from the non-secure software architecture. Section 6 concludes this paper.

2. Related work

Related work addresses security issues in software requirements and architectures. The security requirements for making a system secure are specified on the basis of security constraints, which are derived from the result of threat analysis. Threats to a system are identified for each asset [16] that should be protected against misuse cases [1,26] or abuse cases [15,28]. The process of deriving security requirements for a secure application is described in [16], which also shows relationships between security constraints and security functions.

The misuse case model [1,26] is a method for inferring security requirements from misuse cases. Use cases of an application and their misuse cases are modeled using a "threaten" relationship [1], and then the security requirements for a system are defined to protect against or mitigate misuse cases [1,26]. Similarly, the abuse case modeling [15,28] describes the harms to the system in the interaction between a system and actors, exploiting security privileges to find out the causes of harms. The security requirements are represented as security functions [16,27] that are required to be implemented in a secure system.

Security aspects can be plugged into the software architecture for a secure system using the connectors [2,19], which are viewed as the glue binding the components. The customized connectors can contain security services, which are specified using a Module Interconnection Language (MIL) [2]. A connector [19] that supports an access control security service can be specified using an extended Architecture Description Language, XML-based extensible ADL (xADL).

The UML is an industrial standard for object-oriented modeling. There are several efforts to model security aspects using the UML notation [11] or to extend the UML notation for security modeling [12,14]. Security use cases are modeled separately from application use cases using an extension relationship in the use case model, and security objects are modeled separately from application objects in the analysis model [11]. The UML notation is enriched by including new stereotypes relevant to modeling security aspects [12]. The security constraints are specified at the meta-model [14] level from which a system model can be instantiated.

3. Overview of approach

In this paper, by careful separation of concerns, security requirements are modeled separately from the application requirements, and the security requirements are committed if the application requires security services [11]. In the software architecture, the security services are encapsulated in connectors separately from the components that provide the functionality of an application. The connectors perform security operations if the application requires security services.

Security requirements are captured in security use cases and are encapsulated in security objects. Security use cases are extended from application use cases if the security requirement conditions are satisfied. Similarly, security classes are specialized from the application classes if the application needs to be secure.

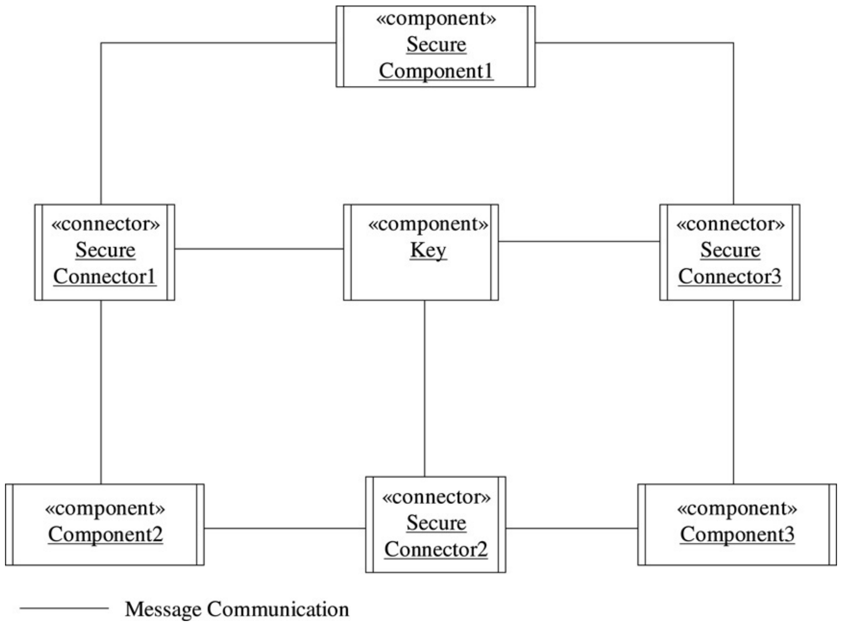
«component» Secure Component1
«connector» Secure Connector1
«component» Key
«connector» Secure Connector3
«component» Component2
«connector» Secure Connector2
«component» Component3
Message Communication

Fig. 1. Secure software architecture with components and connectors.

The software architecture [4,7,23] for concurrent and distributed applications can be designed by means of components and connectors. The components address the functionality of an application, whereas connectors deal with communication between components. Each component defines functionality that is relatively independent of functions provided by other components. A component may request services from other components, or provide services to them through connectors. A connector acts on behalf of components in terms of communication between components, encapsulating the details of inter-component communication.

As a non-secure application evolves to be a secure application, security services are encapsulated in connectors between the components of the software architecture. The original role of connectors in the software architecture is to provide a mechanism for message communication between components. In order to support application security at the software architecture level as the non-secure software architecture evolves into a secure software architecture, the role of connectors has been extended by adding security objects to the connectors, which are then referred to as secure connectors.

The security goals such as confidentiality, integrity, access control, and non-repudiation can be achieved by secure connectors that control interaction between components in the software architecture. Confidentiality services preventing secret information from being disclosed to any unauthorized party can be achieved by secure connectors encapsulating cryptosystems, while integrity services protecting against unauthorized changes to data can be performed by secure connectors using message digest (MD) or message authentication code (MAC) [18]. Similarly, secure connectors are involved in access control by protecting against unauthorized access to resources, and non-repudiation by protecting against one party to a transaction later falsely denying that the transaction occurred. Access control can be implemented using mandatory access control (MAC), while non-repudiation security services may be realized using digital signatures [6].

In this paper, the components in the software architecture will have minimal changes as the non-secure software architecture evolves to the secure software architecture. Some security service encapsulated in a secure connector may require input for performing the security service. In that case, a component connecting to the secure connector is modified to become a secure component that provides the input for the security service.

Fig. 1 depicts the secure software architecture of a distributed application structured into components and their connectors, which has evolved from a non-secure software architecture. The components and connectors are active objects (concurrent objects) that have their own threads of control. In UML, a concurrent object is represented by a rectangle with side bars (e.g., Component2 in Fig. 1). Component1 communicates with Component2 and Component3

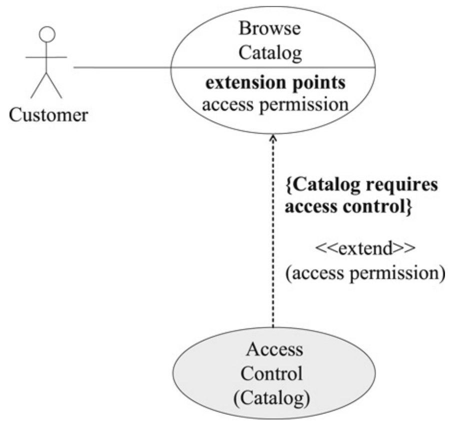

Fig. 2. Browse catalog with access control security use case in electronic commerce system.

via Connector1 and Connector3 respectively. While some components become secure components (e.g., Secure Component1), other components such as Component2 and Component3 remain unchanged. When an application evolves into a secure application, a connector becomes a secure connector if it provides security services (e.g., Secure Connector1). Secure connectors get keys from a key component in order to provide secure message communication between components. However, key management is outside the scope of this paper.

4. Evolution in requirements models for secure applications

When modeling enterprise architectures using UML [3,21], enterprise applications are viewed through multiple views [10] — a functional requirements view is achieved through use case modeling, a static view is achieved using class modeling, and a dynamic view through object communication modeling.

4.1. Use case modeling

As a non-secure application evolves into a secure application, the security use cases are modeled separately from the application use cases. Use case modeling is used to model an application's functional requirements by means of non-secure business use cases and to separately model the security requirements by means of security use cases [11]. When the application requires security services, the security use cases are extended from the non-secure business use cases at extension points. An extension point [3,21] is a location where a use case extends another use case if the proper condition holds. The business use cases specify the functionality of applications. They provide extension points where a security use case extends an application use case, if the appropriate security requirement condition holds. The security use cases can have parameters, whose values are passed from the business use cases that they extend.

An example of the evolution of non-secure application use cases to security use cases is shown in a UML use case diagram (Fig. 2). In a use case for Browse Catalog in an electronic commerce system, a customer browses through various WWW catalogs and views various catalog items from a given supplier's catalog. The Browse Catalog business use case is extended to the Access Control security use case, shaded in gray, if the application requires security services. A customer may need permission to access a specific catalog. This security concern is captured in a separate security use case — Access Control (Catalog). Access Control (Catalog), a security use case, extends Browse Catalog, an application use case, at an extension point called Access Permission, if the application needs to control access to catalogs.

The following describes specifications for Browse Catalog application use case and Access Control security use case.

Use Case Name: Browse Catalog
Summary: Customer chooses a catalog to browse.
Actor: Customer
Precondition: System is ready.
Description (Base use case):

1. Customer chooses to view a catalog index.
2. System displays catalog index.
3. User requests to view a specific catalog.
4. <access permission>
5. System reads the catalog and displays it.

Alternatives:
Postcondition: Customer has browsed a catalog.

In the above example, for a non-secure application where the security condition [Catalog Requires Access Control] is false, step 4 in the Browse Catalog use case is a null function. For a secure application where the security condition [Catalog Requires Access Control] is true, step 4 <access permission> in Browse Catalog is replaced by the Access Control extension use case given below:

Use Case Name: Access Control (Object)
Summary: System authorizes customer or supplier access to object.
Dependency: Extension use case for several use cases.
Actor: Customer
Precondition: Customer or supplier is authenticated as a legitimate user.
Description:

1. Using customer or supplier identity information, the system checks whether customer or supplier has permission to access an {object}.
2. If customer has permission, system permits customer or supplier to gain access to the {object}.

Alternatives:

1. If customer does not have permission to access the {object}, the system displays Access Denied message.

Postcondition: System has permitted customer or supplier to gain access to an {object}.

4.2. Static modeling

The static model is used in UML to depict the static structural view of an application by modeling classes. The static model defines the classes in the application, their attributes and relationships between classes. There are three types of relationship among classes: associations, composition/aggregation, and generalization/specialization relationships.

Using a generalization/specialization hierarchy, the static model of a non-secure application, which consists of non-secure application classes, is extended through inheritance to provide security service classes providing security services. Security concerns are separated from business concerns by modeling non-secure application classes separately from security service classes.

Fig. 3 depicts the evolution of Customer Interface and Customer Agent application classes to secure Customer Interface and secure Customer Agent security classes in an electronic commerce system. The Customer Agent acts on behalf of the human customer to assist the Customer who orders items. The Customer Interface and Customer Agent application classes that provide non-secure business services are specialized respectively to secure Customer Interface and secure Customer Agent security classes that provide secure services if the security requirement condition is satisfied — Catalog requires access control. Each class is depicted with a stereotype, e.g., «user interface», «business logic», «security service», or «database wrapper» (Fig. 4), which represents the role played by the class in the application.

4.3. Dynamic modeling

The communication model is used to depict the objects that participate in each use case, and the sequence of messages passed between them. Once the use cases have been determined, the corresponding communication diagrams can be developed.

As described in Section 4.1, security concerns can be addressed in the use case model by extending the non-secure base use case with a secure extension use case; the secure extension use case is executed if the security condition is

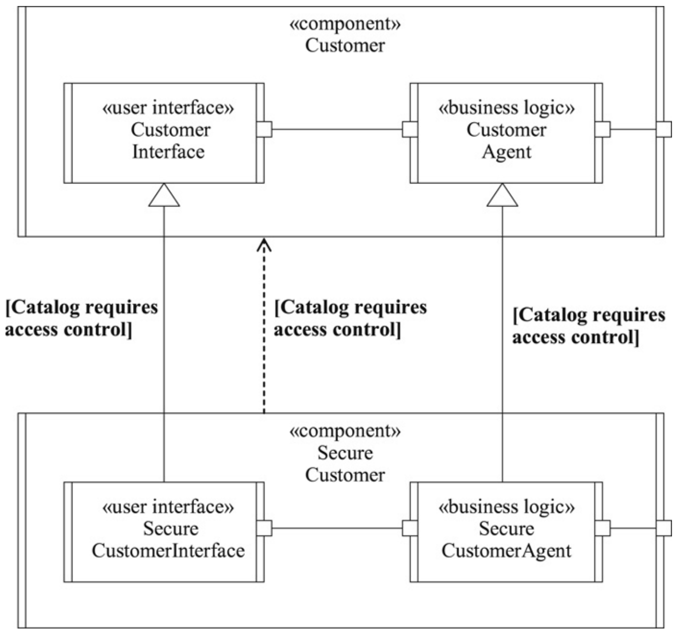

Fig. 3. Generalization/specialization hierarchy for Customer Interface and Customer Agent classes.

true. This concept can be realized in the dynamic communication model by providing a conditional message sequence, which represents a conditional path consisting of objects and messages [10]. Execution of the conditional sequence is guarded by a security requirement condition. Security concerns are separated from the application's business concerns by modeling non-secure application objects separately from security objects [11]. Security concerns captured in separate use cases are encapsulated in separate security objects. If a security requirement condition is true, meaning that the application needs to be secure, non-secure objects communicate with security objects through an alternative message sequence.

Fig. 4 depicts a communication diagram for Browse Catalog with security use case that addresses catalog security when the application requires access control for Catalog (Fig. 2). If Catalog requires access control, Secure Customer Agent requests Access Control Agent to authorize customer access to a specific catalog (message sequence A2.2 through A2.5). Access Control security service is provided by the Access Control Agent and Access Control Server objects. Depending on an organization's access control policy, Access Control Agent authorizes customer access to an object using access control data stored in Access Control Server. The security requirement condition [Catalog requires access control] is used to control whether the security service is accessed or not. The security requirement condition is the same as that used for the extension condition in the use case model (Fig. 2). If the condition is true, the base use case (Browse Catalog in Fig. 2) is extended by the Access Control security use case. The security use case is realized in the communication model by the Secure Customer Agent object in Fig. 4, which, if the security requirement condition holds, directs the customer's request for a catalog to the Access Control Agent, which in turn requests the access control data from the Access Control Server to authorize the customer's request.

5. Evolution of software architectures for secure applications

As a non-secure application evolves to become a secure application, the components in the non-secure application may need to be changed to support the evolution. Some security services encapsulated in secure connectors require input from components, which is used to compute security values. For example, an access control security service relies on customer identity to check the customer's access permission to a specific resource. Other components can remain unchanged in the evolution to a secure software architecture. For instance, confidentiality and integrity security services do not need any input for encrypting/decrypting messages and checking message authentication. The components requiring confidentiality and integrity security services remain unchanged in the evolution to the secure

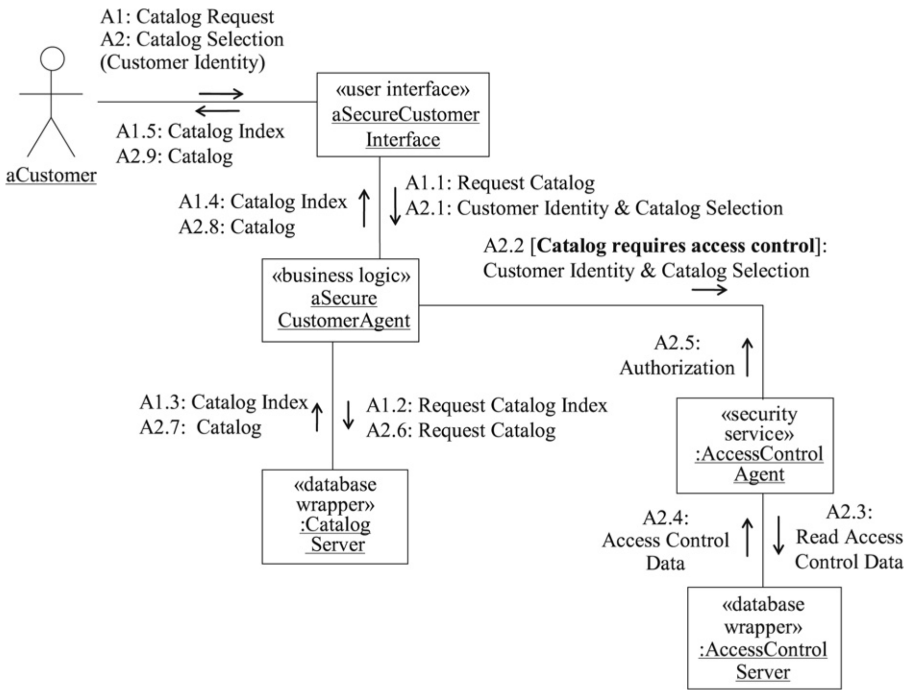
A1: Catalog Request
A2: Catalog Selection (Customer Identity)
A1.5: Catalog Index
A2.9: Catalog
aCustomer
«user interface» aSecureCustomer Interface
A1.1: Request Catalog
A2.1: Customer Identity & Catalog Selection
A1.4: Catalog Index
A2.8: Catalog
A2.2 [Catalog requires access control]: Customer Identity & Catalog Selection
«business logic» aSecure CustomerAgent
A2.5: Authorization
A1.3: Catalog Index
A2.7: Catalog
A1.2: Request Catalog Index
A2.6: Request Catalog
«security service» :AccessControl Agent
«database wrapper» :Catalog Server
A2.4: Access Control Data
A2.3: Read Access Control Data
«database wrapper» :AccessControl Server

Fig. 4. Communication diagram for Browse Catalog with access control security use case.

software architecture. This is because these components do not need to provide input, such as customer identity, to the secure connectors acting on behalf of the components.

5.1. Modeling secure components and connectors

When the application requires security services, a non-secure component in the software architecture must evolve to become a secure component. This is achieved by specializing classes in the non-secure component to become secure classes. A component can be considered a composite or aggregate class that contains the simple classes constituting the component. The simple classes in a component may need to support the evolution of the software architecture for a secure application. Therefore, a non-secure component becomes a secure component as the simple classes contained in the component evolve to become secure classes.

Fig. 3 depicts a secure Customer component in an electronic commerce system that evolves from a non-secure Customer component. The Customer Interface and Customer Agent classes need to provide an access control security service to check the customer access permission to a resource. To provide secure access control, the Customer Interface and Customer Agent classes are specialized to become secure classes. As a result, the composite non-secure Customer component evolves to become the corresponding Secure Customer component.

A non-secure connector evolves to become a secure connector by encapsulating security services such as confidentiality, integrity, access control, and non-repudiation. In the software architecture for a distributed application, connectors encapsulate the details of message communication between components. Connectors can be designed for sending a message to a component (referred to as a sender connector) and receiving a message from a component (referred to as a receiver connector) [9]. To implement a security service, a sender connector and/or a receiver connector contain security classes to provide the security service.

The confidentiality security service can be achieved using a cryptosystem that encrypts a plain message into a cipher message and decrypts the cipher message received into a plain message. A sender connector encapsulates security classes relevant to encryption of a plain message, whereas a receiver connector contains security classes associated with decryption of a cipher message to a plain message. The security objects (instances of security classes) are executed if the application requires a confidentiality security service.

The integrity security service can be fulfilled by a message digest (MD) or message authentication code (MAC) that checks the authenticity of messages communicated between components. A sender connector has security objects

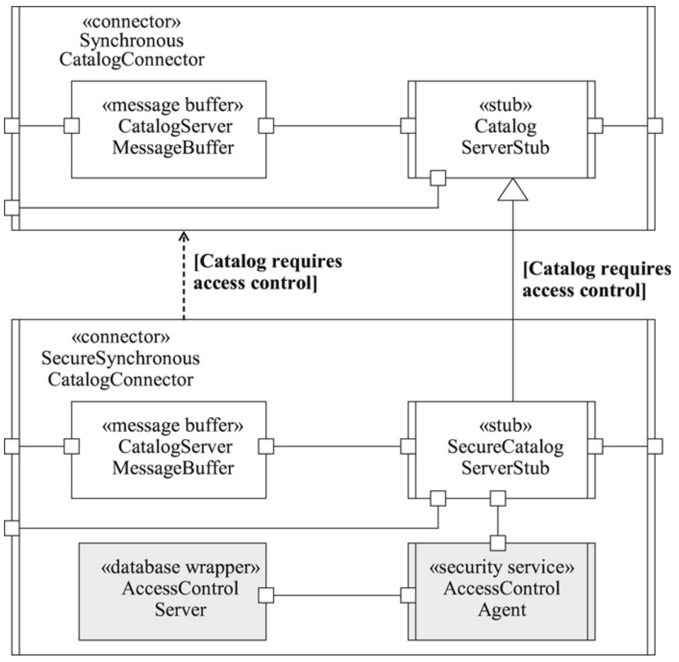

Fig. 5. Evolution of Catalog Connector.

generating a message digest or a message authentication code for a message being sent. The message digest (or message authentication code) is sent to the destination, along with the original message. A receiver connector is supported by security objects, which check the integrity of the received message by means of comparing the received message digest (or message authentication code) with the message digest (or message authentication code) generated by the receiver using the received message. These security objects are performed if the application requires integrity security service.

The non-repudiation security service can be realized using digital signature that signs a message with the signature of the message originator. A sender connector contains security objects signing a message being sent with the originator's signature, whereas a receiver connector encapsulates security objects that prove the authenticity of the received signature using a key (e.g., public key of the originator). These security objects in the secure connectors are activated if the application requires a non-repudiation security service.

The access control security service can be implemented using discretionary (DAC), mandatory (MAC) or role-based access control (RBAC) [22], which allows a user to gain access to an object based on the access control policies. A sender connector might not contain security objects, while a receiver connector contains security objects that check a user's access permission to an object. These security objects are executed if the application requires an access control security service.

Continuing with the catalog example (see Fig. 2), Fig. 5 depicts a structure of Catalog connector that controls the access to Catalog. The non-secure Catalog connector is supported by the Catalog Server Message Buffer and Catalog Server Stub, which communicate with each other via ports [3,21]. The secure Catalog connector replaces the Catalog Server Stub with Secure Catalog Server Stub, and contains security objects – Access Control Agent and Access Control Server (Figs. 4 and 5) – if catalog requires access control. In the secure Catalog connector, the Secure Catalog Server Stub has a port through which it communicates with a port of the Access Control Agent in order to check user access permission to a Catalog.

5.2. Modeling secure component interfaces and interconnections

A component has one or more ports through which it interacts with other components [8]. Each port is defined in terms of provided and/or required interfaces. A provided interface of a port specifies the requests that are required by other components. A required interface of a port specifies the requests that this component requires from other

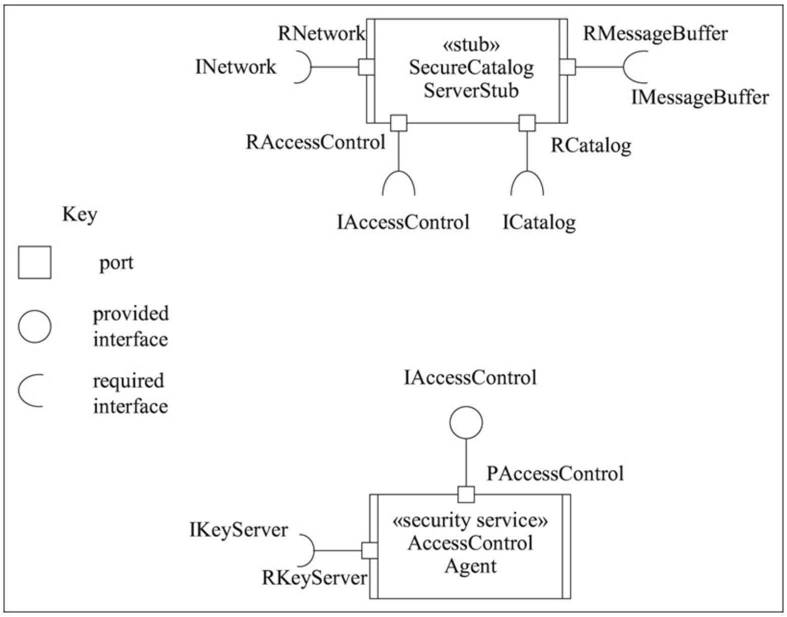

Fig. 6. Ports, provided and required interfaces of Secure Catalog Server Stub and Access Control Agent objects.

components. A provided port is a port that supports a provided interface. A required port is a port that supports a required interface. A complex port is a port that supports both a provided interface and a required interface. A component can support more than one port. In particular, if a component communicates with more than one component, it can use a different port for each component it communicates with.

Fig. 6 depicts Secure Catalog Server Stub and Access Control Agent components with ports as well as provided and required interfaces. The name of a component's required port starts with the letter R to emphasize that the component has a required port. The name of a component's provided port starts with the letter P to emphasize the component has a provided port. The Secure Catalog Server Stub (Fig. 6) has four required ports – RNetwork, RAccessControl, RCatalog, and RMessageBuffer. The Access Control Agent (Fig. 6) has a required port – RKeyServer – and a provided port – PAccessControl. Fig. 7 shows the definition of interfaces of ports of Secure Catalog Server Stub and Access Control Agent.

Fig. 8 depicts a UML communication diagram for the synchronous message communication for controlling the access to the catalog. When a Customer component requests a catalog index, the connectors between the Customer and Catalog Server components deliver the index to the Customer component from the Catalog Server component (message sequence A1 through A1.6). Along with a customer identity, the request for a catalog selected by a customer is sent by a Secure Synchronous Customer connector to a Secure Synchronous Catalog connector (A2 through A2.2), which checks the permission of the customer to gain access to the catalog (A2.3 through A2.5). If the customer has permission to access the catalog, the Secure Catalog Server Stub requests the catalog from the Catalog Server component (A2.6). Fig. 8 describes the collaboration of components and connectors at the software architecture level, whereas Fig. 4 describes the collaboration of objects at the analysis modeling level.

6. Conclusions

This paper has described the evolution of a non-secure application to a secure application by evolving both the requirements model and the software architecture. By careful separation of concerns, security requirements for security services and the software architecture for supporting the security services are modeled separately from application concerns. Security requirements are captured in security use cases and are encapsulated in security objects. The security services are encapsulated in connectors separately from the components that provide the functionality of the application. The security services are invoked if the security requirement conditions are satisfied.

In this paper, the evolution of a non-secure application to a secure application is based around the concept of security requirement conditions. When a security requirement condition is true, the corresponding security

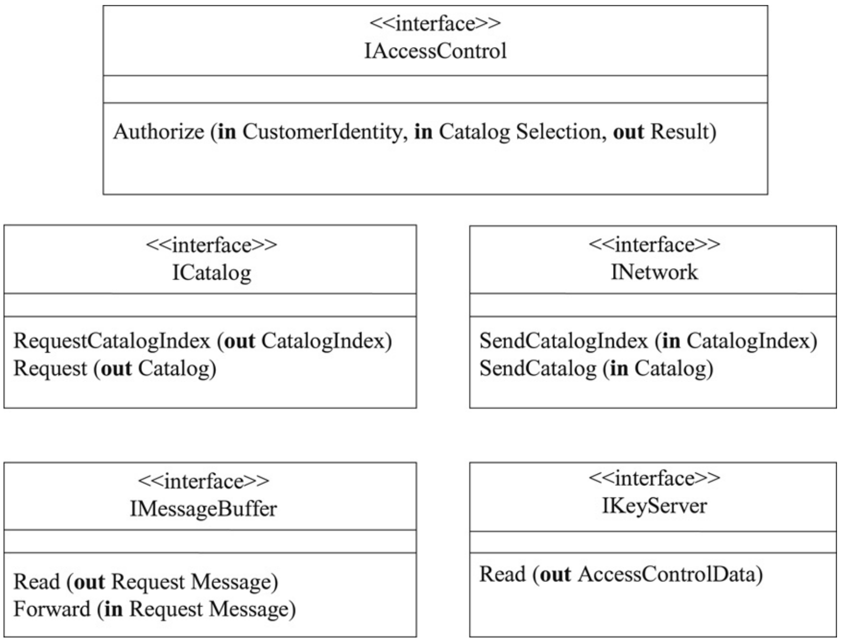
<<interface>>
IAccessControl
Authorize (**in** CustomerIdentity, **in** Catalog Selection, **out** Result)
<<interface>>
ICatalog
RequestCatalogIndex (**out** CatalogIndex)
Request (**out** Catalog)
<<interface>>
INetwork
SendCatalogIndex (**in** CatalogIndex)
SendCatalog (**in** Catalog)
<<interface>>
IMessageBuffer
Read (**out** Request Message)
Forward (**in** Request Message)
<<interface>>
IKeyServer
Read (**out** AccessControlData)

Fig. 7. Interfaces for Secure Catalog Server Stub and Access Control Agent.

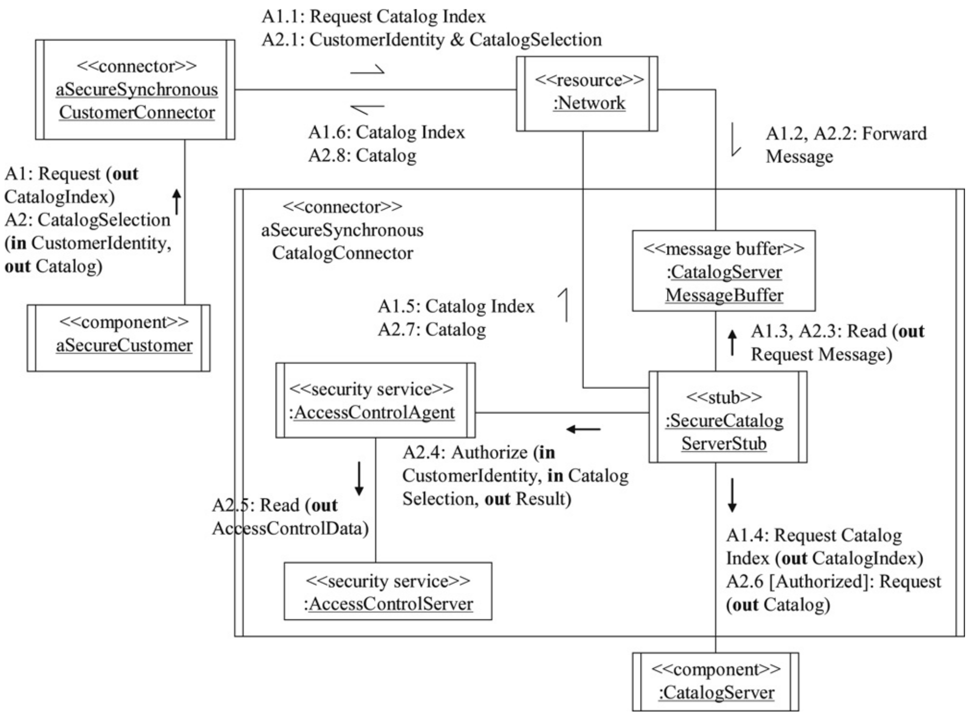
A1.1: Request Catalog Index
A2.1: CustomerIdentity & CatalogSelection
<<connector>>
aSecureSynchronous
CustomerConnector
<<resource>>
:Network
A1.6: Catalog Index
A2.8: Catalog
A1.2, A2.2: Forward Message
A1: Request (**out** CatalogIndex)
A2: CatalogSelection (**in** CustomerIdentity, **out** Catalog)
<<component>>
aSecureCustomer
<<connector>>
aSecureSynchronous
CatalogConnector
<<message buffer>>
:CatalogServer
MessageBuffer
A1.5: Catalog Index
A2.7: Catalog
A1.3, A2.3: Read (**out** Request Message)
<<security service>>
:AccessControlAgent
<<stub>>
:SecureCatalog
ServerStub
A2.4: Authorize (**in** CustomerIdentity, **in** Catalog Selection, **out** Result)
A2.5: Read (**out** AccessControlData)
A1.4: Request Catalog Index (**out** CatalogIndex)
A2.6 [Authorized]: Request (**out** Catalog)
<<security service>>
:AccessControlServer
<<component>>
:CatalogServer

Fig. 8. Communication diagram for secure software architecture for E-Commerce system.

requirement is achieved using a security use case and security objects to provide the security service. In addition, when the security requirement condition holds, the corresponding secure connector in the software architecture fulfills the security services encapsulated in the connector. By separating application concerns from security concerns, the evolution from a non-secure application to a secure application can be achieved with less impact on the application.

Because the approach is built on distributed software architectures using components and connectors, which can be deployed to highly distributed configurations [7], the approach described in this paper is scaleable to large enterprise applications. Validation of the approach is provided through case studies. The non-secure version of the requirements model and software architecture for the E-Commerce system is described in detail in [8]. The secure version of the E-Commerce system described in [11,24] and in this paper. Further validation would necessitate the implementation and testing of the secure application.

The secure modeling approach described in this paper can be further extended to address threat modeling in an application. Threat modeling is used to identify the assets of an application that could be harmed if the application is not secure. The security requirements for protecting the assets can be derived from the threat model for the application. Modeling threats to assets in an application is a starting point for evolving to a secure application. This approach can also be supported by a method and supporting tool that checks the consistency between the requirements model and software architecture for the evolved secure application. Future work involves investigating aspect-oriented modeling for separating business and security concerns. In addition, the secure requirements model and secure software architecture model for a secure application can be analyzed to check the dynamic behavior [5,17, 25] of the system using executable models such as Rational Rose Real-time [20], or Colored Petri Nets [13], or by implementation using a high level programming language.

References

[1] I. Alexander, Misuse cases: Use cases with hostile intent, IEEE Software 20 (1) (2003) 58–66.
[2] C. Bidan, V. Issarny, Security benefits from software architecture, in: Proceedings of the 2nd International Conference on Coordination Languages and Models, 1997, pp. 64–80.
[3] G. Booch, J. Rumbaugh, I. Jacobson, The Unified Modeling Language User Guide, 2nd ed., Addison-Wesley, 2005.
[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, Pattern Oriented Software Architecture: A System of Patterns, John Wiley & Sons, 1996.
[5] Y. Deng, J. Wang, J.J.P. Tsai, K. Beznosov, An approach for modeling and analysis of security system architectures, IEEE Transactions on Knowledge and Data Engineering 15 (5) (2003) 1099–1119.
[6] W. Ford, M.S. Baum, Secure Electronic Commerce: Building the Infrastructure for Digital Signatures and Encryption, Prentice-Hall PRT, 1997.
[7] H. Gomaa, Designing Concurrent, Distributed, and Real-Time Applications with UML, Addison-Wesley, 2000.
[8] H. Gomaa, Designing Software Product Lines with UML, Addison-Wesley, 2005.
[9] H. Gomaa, D. Menasce, E. Shin, Reusable component interconnection patterns for distributed software architectures, in: Proceedings ACM Symposium on Software Reusability, May 2001, ACM Press, Toronto, Canada, 2001, pp. 69–77.
[10] H. Gomaa, M.E. Shin, Multiple-view meta-modeling of software product lines, in: The 8th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS 2002, December 2002, Maryland.
[11] H. Gomaa, M.E. Shin, Modeling complex systems by separating application and security concerns, in: The 9th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS 2004, April 2004, Italy.
[12] J. Jürjens, UMLsec: Extending UML for secure systems development, UML 2002, September 30–October 4, 2002, Dresden.
[13] L.M. Kristensen, S. Christensen, K. Jensen, The practitioner's guide to colored Petri nets, International Journal STTT 2 (1998) 98–132.
[14] T. Lodderstedt, D. Basin, J. Doser, SecureUML: A UML-based modeling language for model-driven security, in: The 5th International Conference on the Unified Modeling Language, September 30–October 4, Dresden, Germany, 2002.
[15] J. McDermott, C. Fox, Using abuse case models for security requirements analysis, in: Proc. of the 15th Annual Computer Security Application Conference, 1999, pp. 55–66.
[16] J.D. Moffett, C.B. Haley, B. Nuseibeh, Core Security Requirements Artefacts, Technical Report, The Open University, UK, 2004.
[17] R. Pettit, H. Gomaa, Modeling behavioral design patterns of concurrent objects, in: Proc. International Conference on Software Engineering, May 2006, Shanghai, China.
[18] C.P. Pfleeger, S.L. Pfleeger, Security in Computing, 3rd ed., Prentice-Hall, Inc. 2002.
[19] J. Ren, R. Taylor, P. Dourish, D. Redmiles, Towards an architectural treatment of software security: A connector-centric approach, in: Software Engineering for Secure Systems, SESS05, ACM SIGSOFT Software Engineering Notes 30 (4) (2005).
[20] Rose Real Time, http://www-306.ibm.com/software/awdtools/developer/rose/, IBM, 2005.
[21] J. Rumbaugh, G. Booch, I. Jacobson, The Unified Modeling Language Reference Manual, 2nd ed., Addison-Wesley, 2004.
[22] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, C.E. Youman, Role-based access control models, IEEE Computer 29 (2) (1996) 38–47.
[23] M. Shaw, D. Garlan, Software Architecture: Perspectives on an Emerging Discipline, Prentice-Hall, 1996.
[24] M.E. Shin, Evolution in multiple-view models in software product families, Ph.D. Dissertation, George Mason University, Fairfax, VA, 2002.
[25] M.E. Shin, A. Levis, L. Wagenhals, D. Kim, Analyzing dynamic behavior of large-scale system through model transformation, International Journal of Software Engineering and Knowledge Engineering 15 (1) (2005) 35–60.
[26] G. Sindre, A.L. Opdahl, Eliciting security requirements with misuse cases, Requirements Engineering 10 (1) (2005) 34–44.
[27] I. Sommerville, Software Engineering, 7th ed., Addison-Wesley, 2004.
[28] T. Srivatanakul, J.A. Clark, F. Polack, Writing effective security abuse cases, Technical Report YCS-2004-375, University of York, 2004.